

Running example problems in BOUT++

M.V. Umansky

**Presented at BOUT++ Workshop
LLNL, Livermore, CA
Sep 3-6, 2013**



This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Security, LLC, Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-PRES-643313

Test problems are essential for verification, also helpful for new users learning the code

- Test problems can be used for verification testing
- Can promptly point to bugs introduced in the code
- Provide simple fast demonstrations how to use the code
- Provide good starting point for developing a new physics application
- Suite of test problems is a part of code distribution, including
 - Gravitational interchange instability
 - Resistive drift instability
 - Shear-Alfven wave
 - Many others...
- Most of these tests can be run automatically by existing scripts

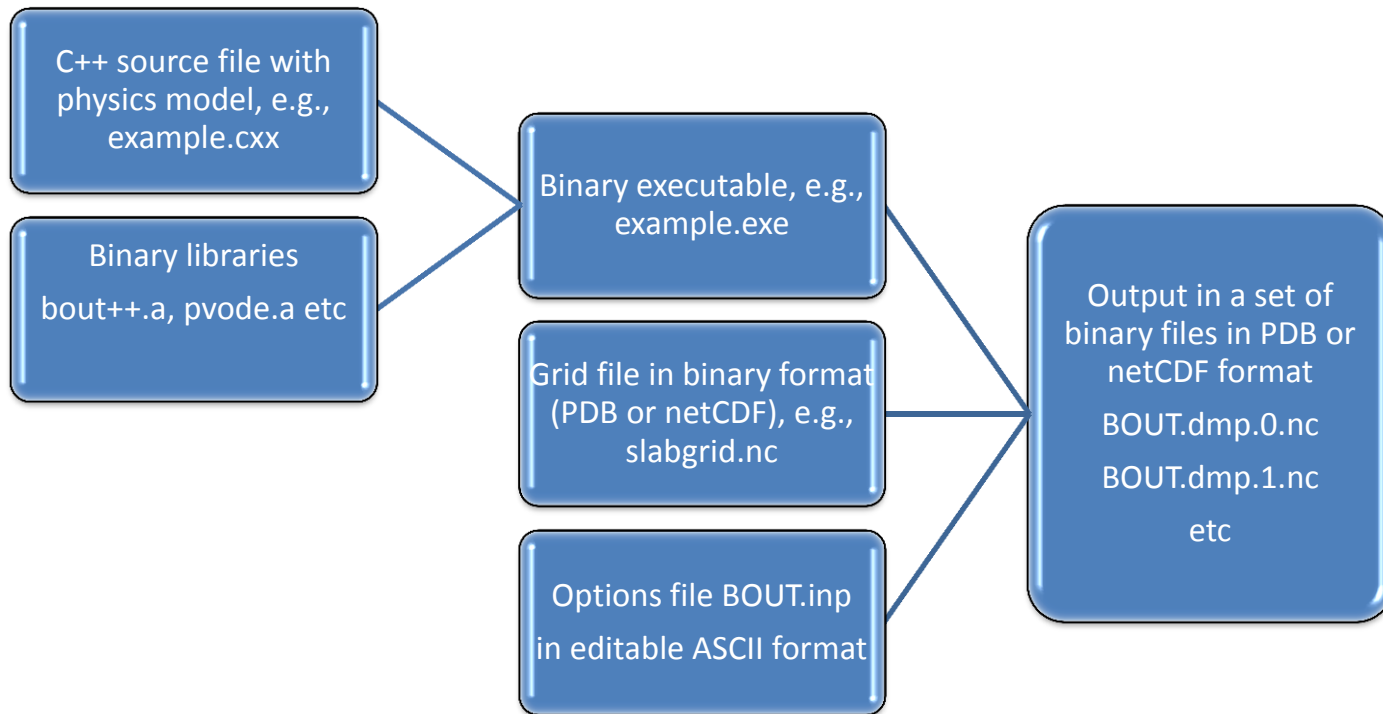
Test problems are essential for verification, also helpful for new users learning the code

- Test problems can be used for verification testing
- Can promptly point to bugs introduced in the code
- Provide simple fast demonstrations how to use the code
- Provide good starting point for developing a new physics application
- Suite of test problems is a part of code distribution, including
 - Gravitational interchange instability
 - Resistive drift instability
 - Shear-Alfven wave
 - Many others...

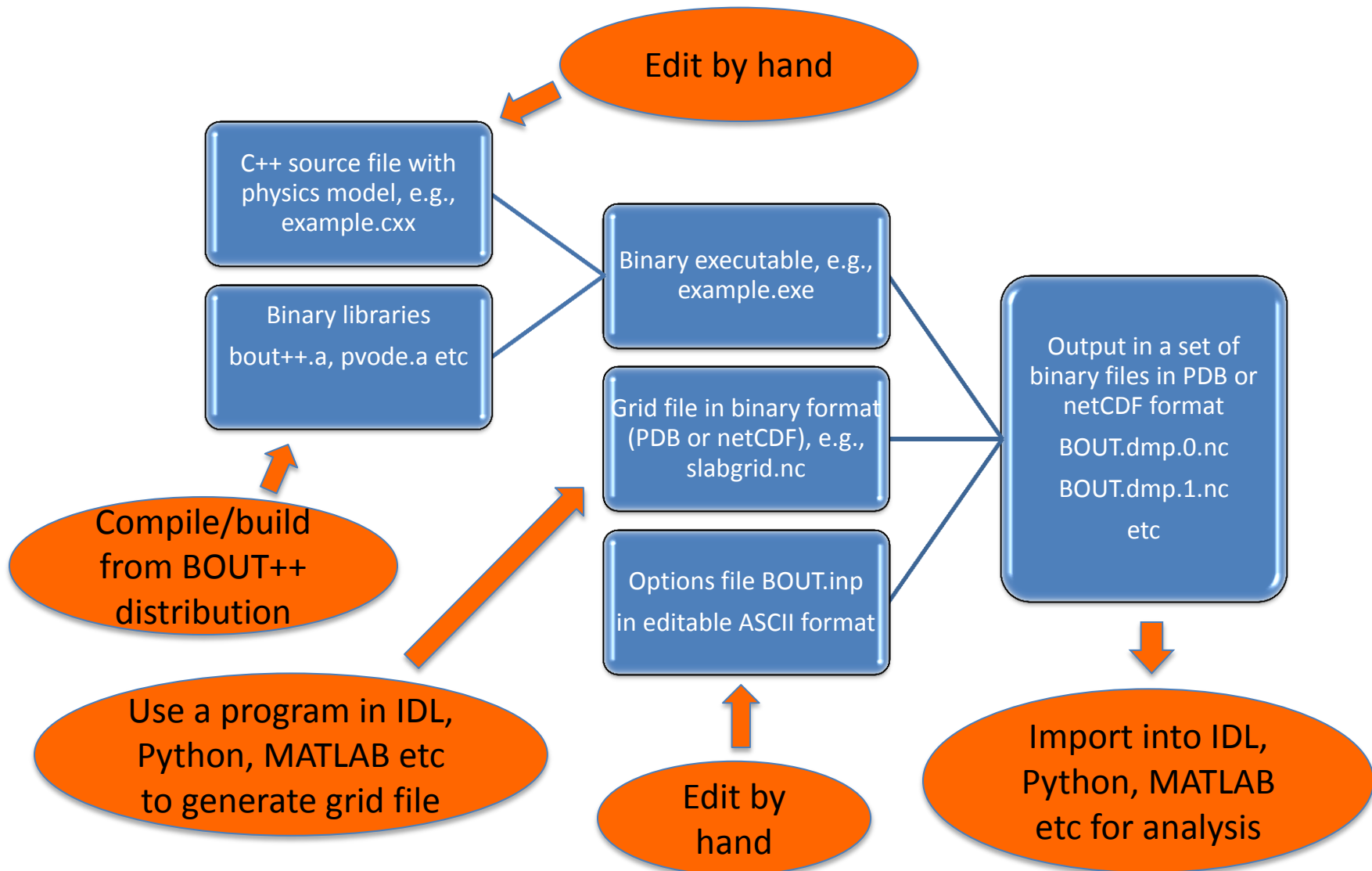
```
> ls examples/
advdiff          elm-pb           laplace-dae      shear-alfven-wave  test-invpar        test-smooth
advdiff2         em-drift         makefile         sod-shock          test-io            test-staggered
advect1d         gravity_reduced  monitor         split-operator     test-laplace       test_suite
conducting-wall-mode gyro-gem        non-local_1d    test               test-laplace2      test-wave
conduction       hasegawa-wakatani  orszag-tang    test-cyclic        test-nonuniform    uedge-benchmark
d3d-119919       hazeltine_4field  rayleigh-taylor test-delp2         test-petsc_laplace
d3d-129131       interchange-instability rb-tokamak     test-fieldfactory  test-petsc_laplace_MAST-grid
dalf3           jorek-compare    README         test-gyro          test-precon
drift-instability lapd-drift       reconnect-2field test-initial        test-simple-diffusion
> █
```

- Most of these tests can be run automatically by existing scripts

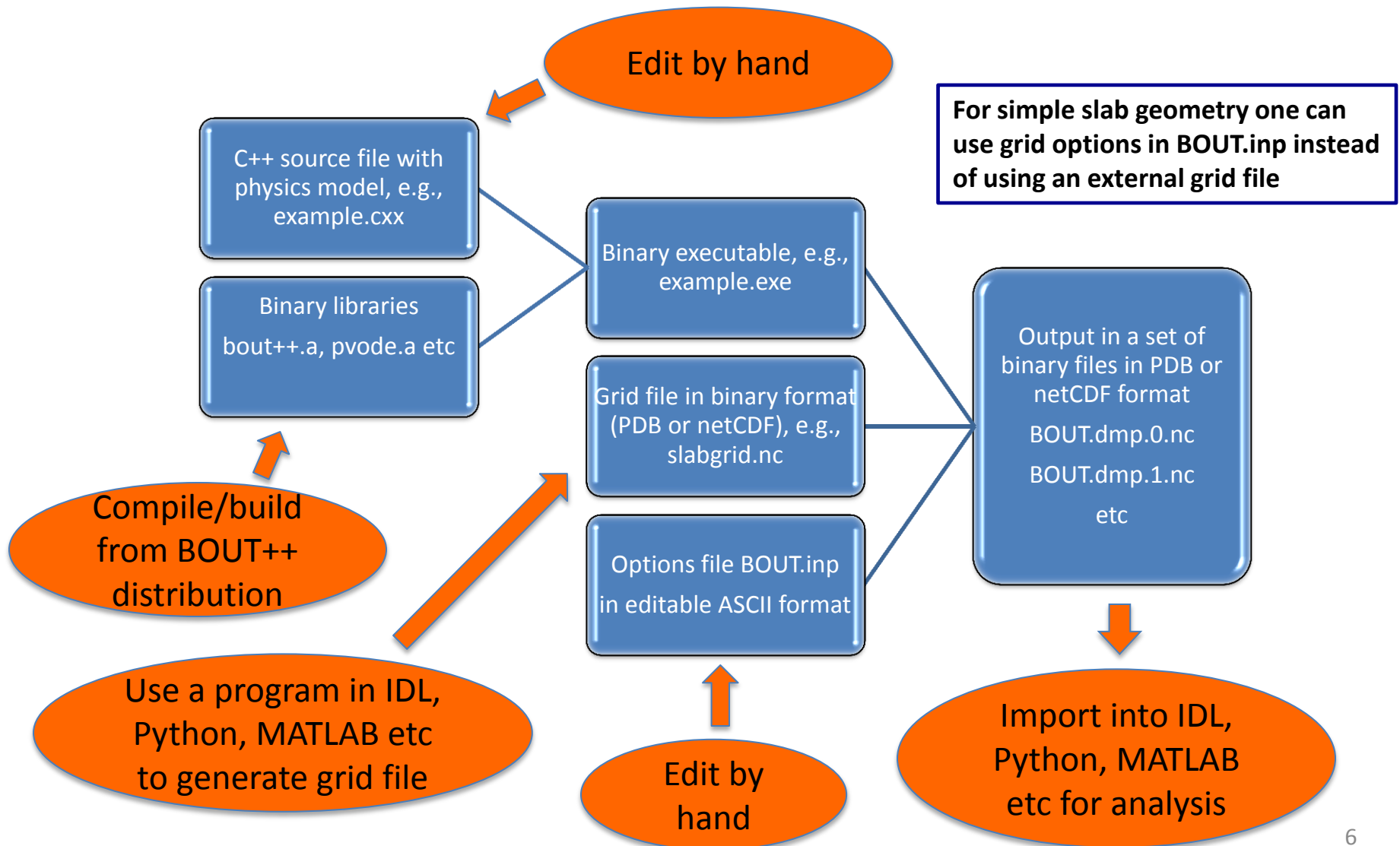
Several steps are involved in setting up and running example cases in BOUT++



Several steps are involved in setting up and running example cases in BOUT++



Several steps are involved in setting up and running example cases in BOUT++



Physics model for gravitational flute instability

From “*Plasma physics and controlled fusion*” by F. F. Chen

To find the growth rate, we can perform the usual linearized wave analysis for waves propagating in the y direction; $\mathbf{k} = k \hat{\mathbf{y}}$. The perturbed ion equation of motion is

$$M(n_0 + n_1) \left[\frac{\partial}{\partial t} (\mathbf{v}_0 + \mathbf{v}_1) + (\mathbf{v}_0 + \mathbf{v}_1) \cdot \nabla (\mathbf{v}_0 + \mathbf{v}_1) \right] = e(n_0 + n_1) [\mathbf{E}_1 + (\mathbf{v}_0 + \mathbf{v}_1) \times \mathbf{B}_0] + M(n_0 + n_1) \mathbf{g} \quad [6-38]$$

We now multiply Eq. [6-36] by $1 + (n_1/n_0)$ to obtain

$$M(n_0 + n_1) (\mathbf{v}_0 \cdot \nabla) \mathbf{v}_0 = e(n_0 + n_1) \mathbf{v}_0 \times \mathbf{B}_0 + M(n_0 + n_1) \mathbf{g} \quad [6-39]$$

Subtracting this from Eq. [6-38] and neglecting second-order terms, we have

$$M n_0 \left[\frac{\partial \mathbf{v}_1}{\partial t} + (\mathbf{v}_0 \cdot \nabla) \mathbf{v}_1 \right] = e n_0 (\mathbf{E}_1 + \mathbf{v}_1 \times \mathbf{B}_0) \quad [6-40]$$

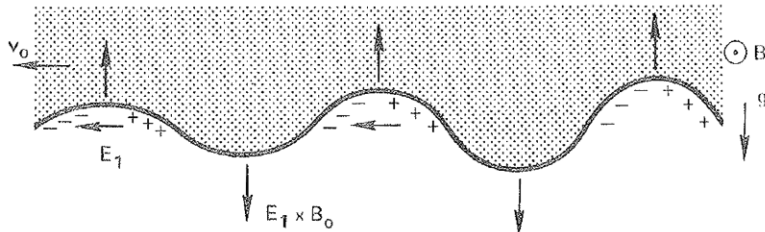


FIGURE 6-11 Physical mechanism of the gravitational instability.

Physics equations in BOUT

$$\frac{\partial \tilde{v}}{\partial t} = 2W_{ci} b_0 \times k \cdot \nabla \tilde{P}$$

$$\frac{\partial \tilde{N}_i}{\partial t} = -\tilde{V}_E \cdot \nabla N_{i0}$$

$$\tilde{v} \equiv N_{i0} \nabla^2 \tilde{f}$$

$$\tilde{P} = T_0 \tilde{N}_i$$

- Equations for perturbations
- Assumed equilibrium terms are in balance
- Vorticity equation combines current continuity and perpendicular momentum
- Certain ordering assumptions built in

In the C++ physics model source file equations are encoded in readable form

Source code for this physics model in
examples/interchange-instability/
2fluid.cxx

Setting up physics
model in the code

$$\frac{\partial \tilde{N}_i}{\partial t} = -\tilde{V}_E \cdot \nabla N_{i0} \longrightarrow$$

$$\tilde{V} \equiv N_{i0} \nabla^2 \tilde{\psi}$$

$$\tilde{P} = T_0 \tilde{N}_i$$

$$\frac{\partial \tilde{V}}{\partial t} = 2W_{ci} b_0 \times k \cdot \nabla \tilde{P} \longrightarrow$$

```
File Edit Options Buffers Tools C++ Help

// Set boundary condition on jpar
jpar.applyBoundary();

// Need to communicate jpar
mesh->communicate(jpar);

Ve = Vi - jpar/Ni0;
Ajpar = Ve;
} else {
    Ve = Ajpar + Apar;
    jpar = Ni0*(Vi - Ve);
}

// DENSITY EQUATION
ddt(Ni) = 0.0;
if(evolve_ni) {
    ddt(Ni) -= vE_Grad(Ni0, phi);

    /*
    ddt(Ni) -= vE_Grad(Ni, phi0) + vE_Grad(Ni0, phi) + vE_Grad(Ni, phi);
    ddt(Ni) -= Vpar_Grad_par(Vi, Ni0) + Vpar_Grad_par(Vi0, Ni) + Vpar_Grad_par(Vi, Ni);
    ddt(Ni) -= Ni0*Div_par(Vi) + Ni*Div_par(Vi0) + Ni*Div_par(Vi);
    ddt(Ni) += Div_par(jpar);
    ddt(Ni) += 2.0*V_dot_Grad(b0xcv, pe);
    ddt(Ni) -= 2.0*(Ni0*V_dot_Grad(b0xcv, phi) + Ni*V_dot_Grad(b0xcv, phi0) + Ni*V_dot_Grad(b0xcv, phi));
    */
}
```

```
File Edit Options Buffers Tools C++ Help

// ELECTRON TEMPERATURE
ddt(Te) = 0.0;
if(evolve_te) {
    ddt(Te) -= vE_Grad(Te0, phi) + vE_Grad(Te, phi0) + vE_Grad(Te, phi);
    ddt(Te) -= Vpar_Grad_par(Ve, Te0) + Vpar_Grad_par(Ve0, Te) + Vpar_Grad_par(Ve, Te);
    ddt(Te) += 1.333*(Ti0*V_dot_Grad(b0xcv, pe)/Ni0 - Vi_dot_Grad(b0xcv, phi));
    ddt(Te) += 3.333*Te0*V_dot_Grad(b0xcv, Te);
    ddt(Te) += (0.6666667/Ni0)*Div_par_K_Grad_par(kapa_Te, Te);
}

// ION TEMPERATURE
ddt(Ti) = 0.0;
if(evolve_ti) {
    ddt(Ti) -= vE_Grad(Ti0, phi) + vE_Grad(Ti, phi0) + vE_Grad(Ti, phi);
    ddt(Ti) -= Vpar_Grad_par(Vi, Ti0) + Vpar_Grad_par(Vi0, Ti) + Vpar_Grad_par(Vi, Ti);
    ddt(Ti) += 1.333*(Ti0*V_dot_Grad(b0xcv, pe)/Ni0 - Ti*V_dot_Grad(b0xcv, phi));
    ddt(Ti) -= 3.333*Ti0*V_dot_Grad(b0xcv, Ti);
    ddt(Ti) += (0.6666667/Ni0)*Div_par_K_Grad_par(kapa_Ti, Ti);
}

// VORTICITY
ddt(rho) = 0.0;
if(evolve_rho) {
    /*
    ddt(rho) -= vE_Grad(rho0, phi) + vE_Grad(rho, phi0) + vE_Grad(rho, phi);
    ddt(rho) -= Vpar_Grad_par(Vi, rho0) + Vpar_Grad_par(Vi0, rho) + Vpar_Grad_par(Vi, rho);
    */

    // ddt(rho) += 2.0*Bxy*V_dot_Grad(b0xcv, pei);
    // ddt(rho) += 2.0*mesh->Bxy*b0xcv*Grad(pe);
    // ddt(rho) += Bxy*Bxy*Div_par(jpar, CELL_CENTRE);
}
```


Grid file can be generated with an IDL routine

In `/tools/slab/slab.pro`: Generating arrays of geometry and plasma parameters and saving them in a binary file (NetCDF or PDB)

```
Slab geometry grid generator
;
; Optional keywords:
; =====
;
; output = Set output file name
; thin   = Use thin radial box approximation
;         so Bpxy = constant, but gradient is non-zero
;
; ni      = Ion density in 10^20 m^-3
; Ti      = ion temperature in eV
; Te      = electron temperature in eV
;
; Rmaj    = Major radius [meters]
; rminor  = Minor radius [m]
; dr      = Radial width of box [m]
; r_wid   = Radial extent, normalised to gyro-radius r_wid = dr / rho_i
;
; q       = Safety factor q = r*Bt/(R*Bp) at middle of box
; dq      = Change in q. Will go from q-dq/2 to q+dq/2
;
; L_T     = Temperature length scale [m]
; L_n     = Density length scale [m]
; eta_i   = Ratio of density to temp. length scales eta = L_n / L_T

PRO slab, output=output, thin=thin, nx=nx, ny=ny, $
    ni=ni, Ti=Ti, Te=Te, $
    Rmaj=Rmaj, rminor=rminor, dr=dr, $
    r_wid=r_wid, $
    q=q, dq=dq, $
    L_T=L_T, L_n=L_n, eta_i=eta_i

if not keyword_set(NX) then nx = 68 ; Radial grid points
if not keyword_set(NY) then ny = 32 ; Poloidal (parallel) grid points
```

```
File Edit Options Buffers Tools Debug IDLWAVE Help

yup_xout = [-1]
ydown_xin = [0]
ydown_xout = [-1]
nrad = [nx]
npol = [ny]

; =====
PRINT, "Writing grid to file "+output

handle = file_open(output, /CREATE)
; Size of the grid

s = file_write(handle, "nx", nx)
s = file_write(handle, "ny", ny)

; Topology for original scheme
s = file_write(handle, "ixseps1", ixseps1)
s = file_write(handle, "ixseps2", ixseps2)
s = file_write(handle, "jyseps1_1", jyseps1_1)
s = file_write(handle, "jyseps1_2", jyseps1_2)
s = file_write(handle, "jyseps2_1", jyseps2_1)
s = file_write(handle, "jyseps2_2", jyseps2_2)
s = file_write(handle, "ny_inner", ny_inner);

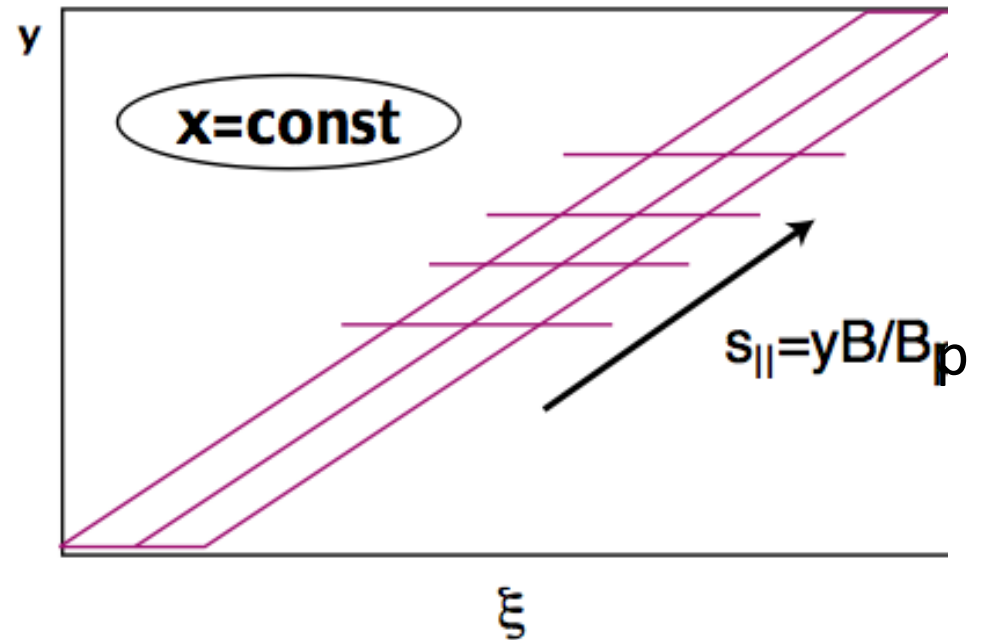
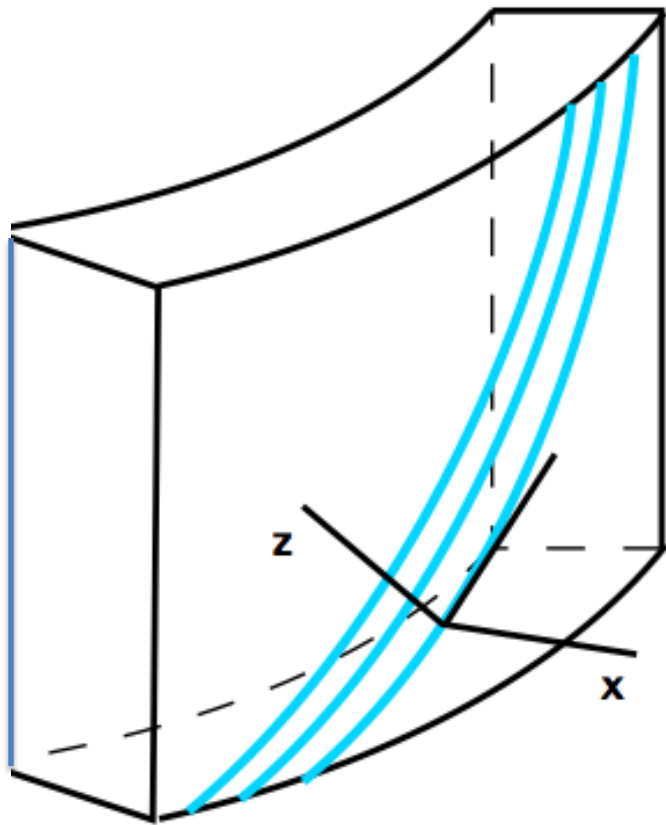
; Grid spacing

s = file_write(handle, "dx", dx)
s = file_write(handle, "dy", dy)

s = file_write(handle, "ShiftAngle", ShiftAngle)
s = file_write(handle, "zShift", zShift)
s = file_write(handle, "pol_angle", pol_angle)
s = file_write(handle, "ShiftTorsion", dqdpsi)

slab.pro (IDLWAVE Abbrev Fill) --L239--70%
```

And this is what the grid data looks like...



Code run is managed with a shell script

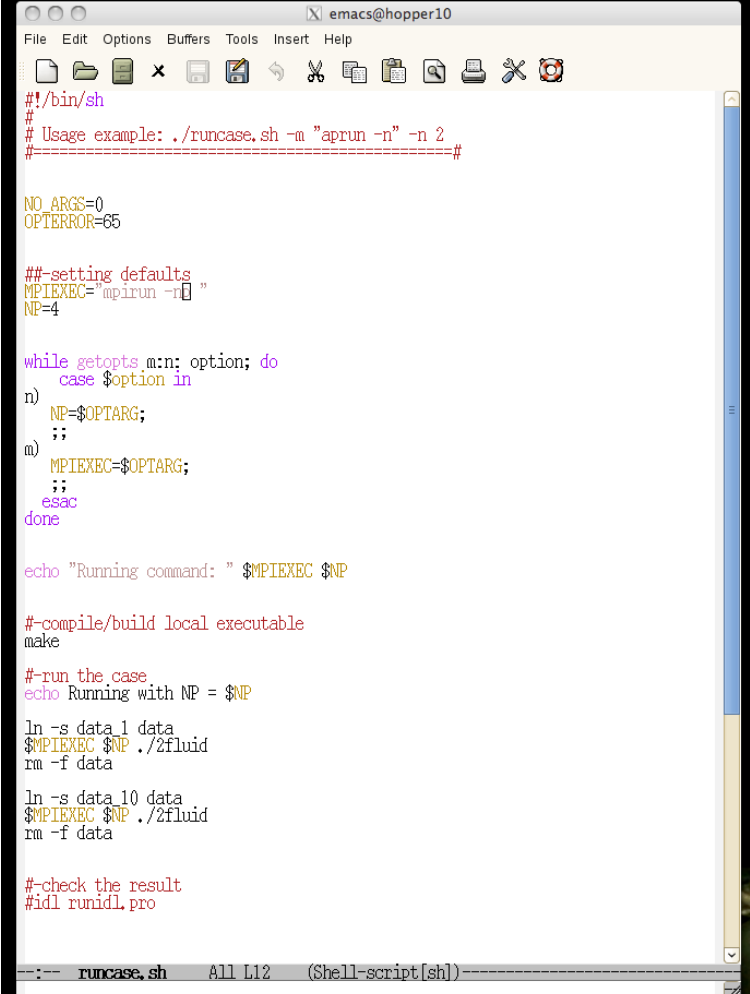
In examples/interchange-instability/runcase.sh

- > `cd <>/examples/interchange-instability`
- > `source runcase.sh`
- ...code running...
- > `idl runidl.pro` => *produces plots*

Two cases, $R=1$ and $R=10$,
are run by the script,
for different values of
radius of curvature

$R=1$ m

$R=10$ m



```
#!/bin/sh
# Usage example: ./runcase.sh -m "aprun -n" -n 2

NO_ARGS=0
OPTERROR=65

##-setting defaults
MPIEXEC="mpirun -n"
NP=4

while getopts m:n: option; do
  case $option in
    n)
      NP=$OPTARG;
      ;;
    m)
      MPIEXEC=$OPTARG;
      ;;
    esac
  done

  echo "Running command: " $MPIEXEC $NP

  #-compile/build local executable
  make

  #-run the case
  echo Running with NP = $NP

  ln -s data_1 data
  $MPIEXEC $NP ./2fluid
  rm -f data

  ln -s data_10 data
  $MPIEXEC $NP ./2fluid
  rm -f data

  #-check the result
  #idl runidl.pro
```

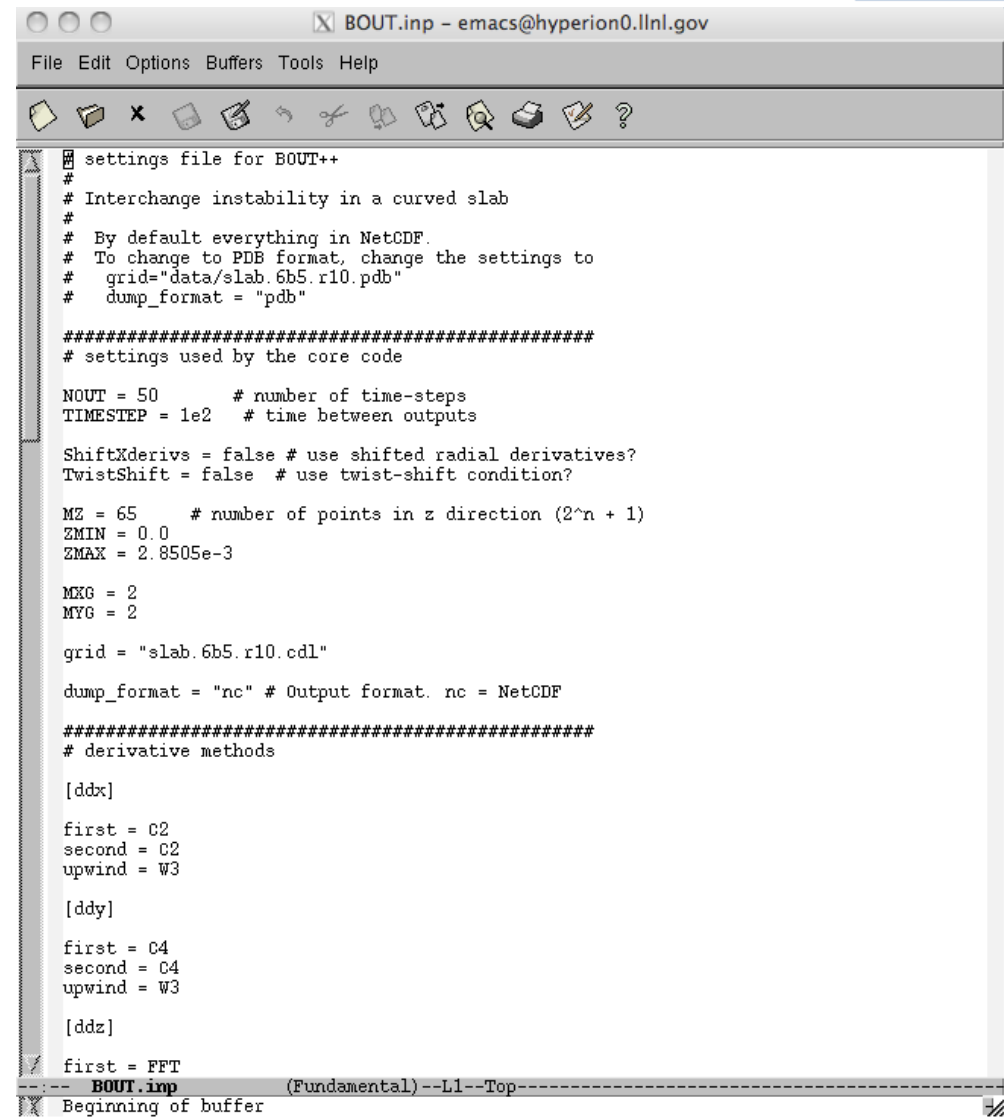
Options file BOUT.inp is used to change basic physics and numerical options without recompiling

In examples/interchange-
instability/data_1/BOUT.inp
data_10/BOUT.inp

Only difference in using different
geometry files:

grid = "slab.6b5.r1.cdl"

grid = "slab.6b5.r10.cdl"

A screenshot of an Emacs editor window titled "BOUT.inp - emacs@hyperion0.llnl.gov". The window displays the contents of the BOUT.inp file, which is a settings file for BOUT++. The file contains various parameters for the simulation, including the number of time-steps (NOUT = 50), time between outputs (Timestep = 1e2), and the number of points in the z direction (MZ = 65). It also specifies the grid file (grid = "slab.6b5.r10.cdl") and the output format (dump_format = "nc"). The file is divided into sections by comments, such as "settings used by the core code" and "derivative methods". The status bar at the bottom indicates the current buffer is "BOUT.inp" and the position is "(Fundamental)--L1--Top".

```
# settings file for BOUT++
#
# Interchange instability in a curved slab
#
# By default everything in NetCDF.
# To change to PDB format, change the settings to
#   grid="data/slab.6b5.r10.pdb"
#   dump_format = "pdb"
#####
# settings used by the core code
#####
NOUT = 50      # number of time-steps
Timestep = 1e2 # time between outputs

ShiftXderivs = false # use shifted radial derivatives?
TwistShift = false  # use twist-shift condition?

MZ = 65      # number of points in z direction (2^n + 1)
ZMIN = 0.0
ZMAX = 2.8505e-3

MXG = 2
MYG = 2

grid = "slab.6b5.r10.cdl"

dump_format = "nc" # Output format. nc = NetCDF
#####
# derivative methods
#####

[ddx]

first = C2
second = C2
upwind = W3

[ddy]

first = C4
second = C4
upwind = W3

[ddz]

first = FFT
-- BOUT.inp (Fundamental)--L1--Top-----
Beginning of buffer
```

Output is processed with IDL script which produces plots, numerical values etc

In examples/interchange-instability/
runidl.pro

```
runidl.pro - emacs@hyperion0.llnl.gov
File Edit Options Buffers Tools Debug IDLWAVE Help

.run pdb2idl.pro
.run moment_xyzt.pro

d = collect(path="data_1", var="Ni", x=5, y=30)
moment_xyzt, d, rms=nil

d = collect(path="data_10", var="Ni", x=5, y=30)
moment_xyzt, d, rms=nil0
tt = collect(path="data_10", var="t_array")
wci = collect(path="data_10", var="wci")
tt = tt[1:*] / wci

nil = REFORM(nil[0,0,1: *])
nil0 = REFORM(nil0[0,0,1: *])

;-compare with original test results (grendel, 31-jan-2007)
RESTORE, 'orig_test.idl.dat'
error1=max(abs((nilorig-nil)/nilorig)) + max(abs((nil0orig-nil0)/nil0orig))
print, "Deviation from original test result is", error1*1e2, " %"

set_plot, 'PS'
device, file='interchange_inst_test.ps'
safe_colors, /first

xtit='t, s' & ytit='rms <Ni>' & tit='Interchange instability test'
plot, tt, nil, /yl, psym=4, xtit=xtit, ytit=ytit, tit=tit, chars=1.5,col=1, xticks=3
oplot, tt, 1e-4*exp(tt*2.2e5),col=1
oplot, tt, nil0, psym=4,col=1
oplot, tt, 1e-4*exp(tt*6.3e4),col=1, lin=2

oplot, tt, nilorig,psym=7,col=1
oplot, tt, nil0orig, psym=7,col=1

xyouts, 7e-5, 0.15, "R0=10 m",col=1
xyouts, 4e-5, 1e2, "R0=1 m",col=1

; calculate growth rates

nlg0 = MEAN(DERIV(tt, ALOG(nilorig)))
nlg = MEAN(DERIV(tt, ALOG(nil)))

nld = 100. * ABS(nlg0 - nlg) / nlg0

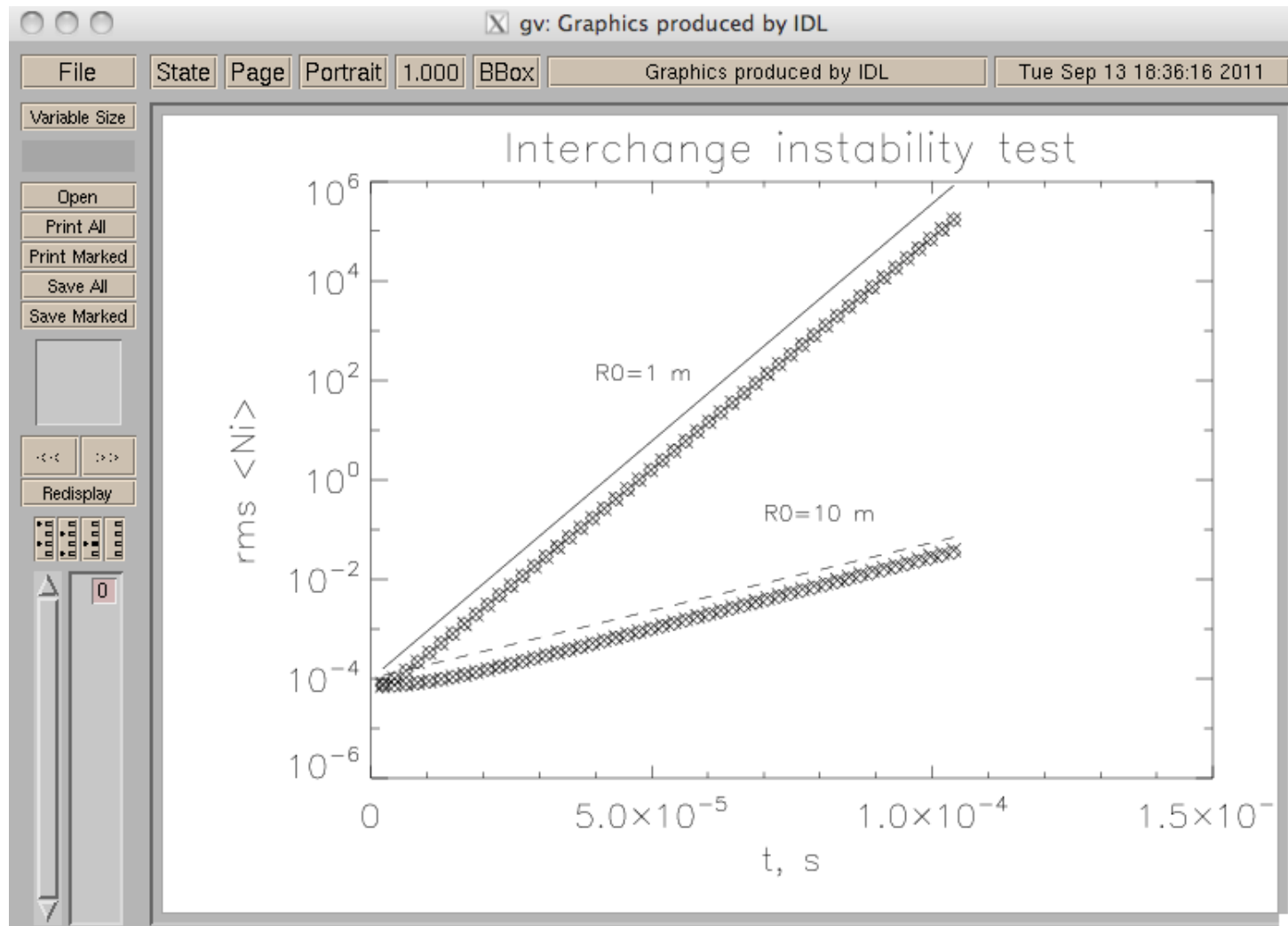
n10g0 = MEAN(DERIV(tt, ALOG(nil0orig)))
n10g = MEAN(DERIV(tt, ALOG(nil0)))

r10d = 100. * ABS(n10g0 - n10g) / n10g0

-- runidl.pro (IDLWAVE Abbrev Fill)--L47--Top-----
Beginning of buffer
```

BOUT++ correctly reproduces analytic answer for the interchange mode growth rate!

Run results plotted in examples/interchange-instability/interchange_inst_test.ps



Suggested exercises for practice

- **Basic**
 - Run the interchange-instability case
 - Run the IDL processing script
 - View the plot

cd <path to BOUT++>/examples/interchange-instability

source runcase.sh

idl runidl.pro

gv interchange_inst_test.ps

- **Intermediate**
 - Read/understand/modify the source file 2fluid.cxx and scripts runcase.sh, runidl.pro
- **Advanced**
 - Try setting up a new simple case from scratch

Downloading and compiling BOUT++ on hopper

- `more /global/homes/u/umansky/BOUT_Workshop_2013/README`
- `module swap PrgEnv-pgi PrgEnv-gnu`
- `module load gcc/4.6.3`
- `module load netcdf/4.1.3`
- `module load fftw`
- `module load idl/bout`
- `module load gv`
- `setenv BOUT_TOP $HOME/BOUT_Workshop_2013/BOUT-2.0/`
- `setenv BW13 /global/homes/u/umansky/BOUT_Workshop_2013`
- `setenv IDL_STARTUP $BW13/idl_startup.pro`
- `mkdir BOUT_Workshop_2013`
- `cd BOUT_Workshop_2013/`
- `git clone https://github.com/boutproject/BOUT-2.0.git`
- `cd BOUT-2.0/`
- `./configure`
- `make`